NISTIR 6150

Molecular Dynamics Simulation of Tethered Chains

Raymond D. Mountain
Physical and Chemical Properties Division

Joseph Hubbard
Biotechnology Division

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Gaithersburg, MD 20899-0001

NIST

NISTIR 6150

Molecular Dynamics Simulation of Tethered Chains

Raymond D. Mountain
Physical and Chemical Properties Division

Joseph Hubbard
Biotechnology Division

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Gaithersburg, MD 20899-0001

April 1998

Molecular Dynamics Simulation of Tethered Chains

Raymond D. Mountain†
Physical and Chemical Properties Division
National Institute of Standards and Technology
Gaithersburg, MD 20899 USA

and

Joseph Hubbard‡
Biotechnology Division
National Institute of Standards and Technology
Gaithersburg, MD 20899 USA

ABSTRACT

A description is provided for a molecular dynamics code that simulates the dynamics of long chain molecules tethered to a smooth surface. The operation of the code is discussed from the point of view of the user. The structure of the input files is described and a discussion on how to change the state of the system (temperature and density/molecule) is provided. The steps needed to change the model are listed. Properties derived from some simulations are provided. A listing of the source code is included in an appendix.

Key words: density profile, energy profile, long chain molecules, molecular dynamics, source listing, united atom model.

1. **Introduction**

The computer simulation method known as molecular dynamics is a powerful means for modeling complex molecular systems. This report describes the structure and operation of a molecular dynamics code that is designed to simulate chain molecules tethered to a surface. In particular, the model explicitly incorporated into the code is one for alkanethiol chains interacting with a metallic (Au) surface.[1] It is known in the literature as Model I. This report will inform the user on the use of this code and will serve as a guide on how to modify it so that other systems can be examined. The source code is available from the authors with the understanding that it is supplied without expressed or implied warranty. Further, it is understood that it is a contribution of the National Institute of Technology and is not subject to copyright. A listing of the code is contained in Appendix 2.

† Email: RMountain@nist.gov
‡ Email: joseph.hubbard@nist.gov

The model is discussed in detail in Section 2. This is done in two steps. First the formal structure of the model is described and then the explicit values of the model parameters are specified.

The code that implements this model is discussed in Section 3. First, the mechanics of running the program for specific conditions is described. Also, constraints on the sort of systems modeled are listed. This discussion of the code includes a brief description of the outputs of the code. Next, the code is examined at a more detailed level. One part of this discussion describes the implementation of the model. Particular attention is paid to the evaluation of the energy, forces, and potential energy and density profiles along with the integration of the equations of motion. It also contains a description of some of the properties that are generated during a simulation run. This description includes the property, how it is evaluated, and where the results are stored. Some sample results are presented to illustrate this discussion.

Section 4 contains some results for three different densities of chains at room temperature to illustrate the types of order possible for these systems.

Section 5 contains a discussion of what would be needed to make changes in several features of the simulation and in the details of the model.

2. **The model**

The properties of linear chain alkanethiol chains forming a (partial) monolayer on a metal surface are estimated using a molecular model. The molecules are represented in terms of a united atom, site-site interaction model for the chains. A "site" is a point where the force is applied. For the model considered here, a site corresponds to a carbon or sulfur atom position. This type of model for linear alkanes has been shown to provide satisfactory descriptions of several properties.[2] However, the parameters for the alkanethiol model are different from the corresponding parameters for alkanes as these are purely empirical models. It is not necessary that the sites correspond to an atomic position.[3] Also, it is necessary to introduce molecule-surface interactions that maintain the monolayer. The interactions are divided into intramolecular terms, intermolecular terms, and surface-atom interaction terms. We now examine each type of interaction.

Intramolecular and intermolecular interactions

The intramolecular interaction,[1] V_{intra}, consists of four parts,

$$V_{intra} = V_2 + V_3 + V_4 + V_5.$$

The parameters in each term depend on the species of the sites entering the interaction although this is not explicitly indicated here in order to avoid cluttering the notation.

There are three species in the model; the thiol group, the CH_2 group, and the CH_3 group. Each group is treated as single "atom" with the appropriate mass, hence the term united atom model. In this report, the term "atom" is to be interpreted as "united-atom". This means that there are several values possible for each of the parameters. The specific values are listed for the parameters in several tables in the following subsection. The V_2 term contains the harmonic stretch interactions between bonded neighbors,

$$V_2 = \frac{1}{2} \sum_{i=1}^{N-1} \gamma_2 (r_{ij} - d_0)^2,$$

where $j = i{+}1$ and N is the number of atoms in a chain. The parameters for the stretch interaction are γ_2 and d_0. The V_3 term contains the harmonic bend interactions that involve adjacent triples of sites,

$$V_3 = \frac{1}{2} \sum_{i=1}^{N-2} \gamma_3 (\theta_{ijk} - \theta_0)^2,$$

$j = i{+}1$, $k = i{+}2$, and θ_{ijk} is the angle at site *j* subtended by sites *i* and *k*. The parameters for the bend interaction are γ_3 and θ_0. The V_4 term contains the torsion interactions that involves adjacent quadruples of sites,[4]

$$V_4 = \sum_{i=1}^{N-3} \sum_{l=0,5} C_l \cos^l(\phi)$$

where ϕ is the dihedral angle between the planes defined by the four sites. The C_l's are the parameters for the torsion interactions. The mechanics of evaluating the three- and four-body energy and forces is discussed in Appendix 1.

The V_5 term is a Lennard-Jones interaction between sites separated by three or more sites,

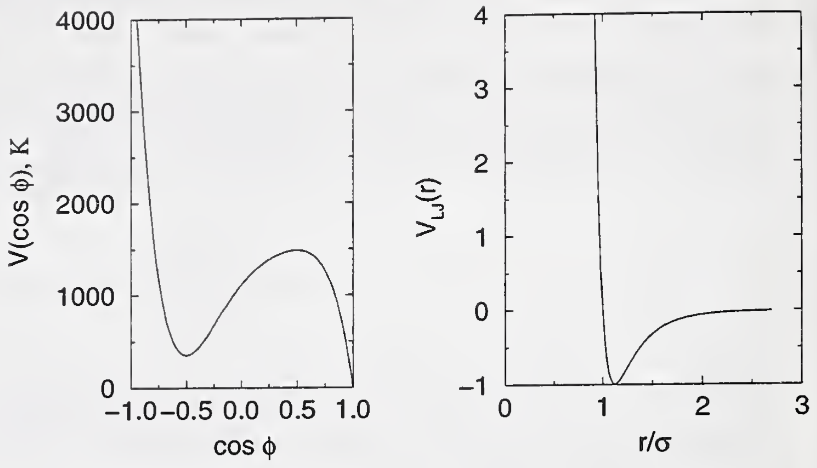
$$V_5 = \sum_{j=1}^{N-4} \sum_{k=j+4}^{N} V_{LJ}(r_{jk}).$$

The Lennard-Jones potential has the form

$$V_{LJ}(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right],$$

with energy depth and range parameters ϵ and σ, respectively.

Figure 1. The torsion potential is shown on the left, in units of kelvins. The parameters are those found in Table 3 below. The Lennard-Jones potential is shown on the right in units of the parameter ϵ.



The torsion and Lennard-Jones potentials are shown in Figure 1.

The intermolecular interactions are determined using the same Lennard-Jones potential that acts between intramolecular sites except that it acts between sites on distinct chain molecules.

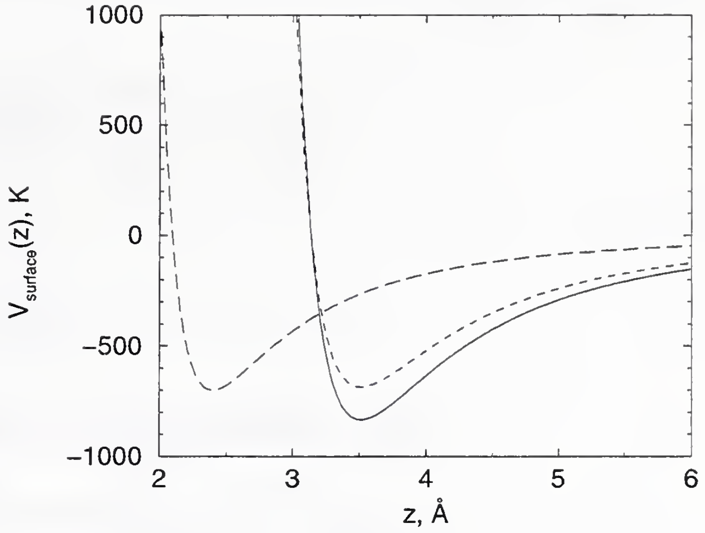
Surface interactions

The surface interaction introduced by Hautman and Klein[1] is a 12-3 surface potential of the form,

$$V_{surface}(z) = \frac{C_{12}}{(z - z_0)^{12}} - \frac{C_3}{(z - z_0)^3}.$$

The interaction of each site in the chain depends on the distance of the site above the surface. The surface is otherwise unstructured. The surface interactions are shown in Figure 2. The introduction of surface structure is straightforward[5] but discussion of it is beyond the scope of this report.

Figure 2. The surface potentials are shown here using the parameters from Table 4. The potential for the CH_3 sites is represented by the solid line, the potential for the CH_2 sites is represented by the short dashed line, and the potential for the thiol sites, *reduced by a factor of 20*, is represented by the long dashed line.



Model parameters, physical units

Next we introduce the potential parameters. First the parameters are assigned physical values and then these values are put into the scaled form used in the code.

Table 1. Stretch parameters in physical units. The CH_3 and CH_2 sites are equivalent for the stretch interaction.

Parameter	S-C	C-C
γ_2, 10^7 K/nm^2	4.529	4.529
d_0, Å	1.82	1.54

Table 2. Bend parameters in physical units.

Parameter	C-C-C	S-C-C
γ_θ, 10^3K/rad^2	62.5	62.5
θ_0, deg	109.5	114.4

There are two sets of values for the thiol group. The $S_I - S_I$ parameters were introduced to obtain the proper spacing of the chains at the surface. The other parameters are intended to provide more realistic interactions between the thiol and the CH_2 and CH_3 groups.

Table 3. The coefficients, C_l, in the torsional potential.[4] Note that there is no site dependence for these coefficients

l	C_l, K
0	1116
1	1462
2	-1578
3	-368
4	3156
5	-3788

Table 4. The parameters for the surface interactions.[1]

Site	C_{12}, 10^7KÅ^{12}	C_3, KÅ^3	z_0, Å
CH_3	3.41	20800	0.860
CH_2	2.80	17100	0.860
S	4.089	180600	0.269

Table 5. Lennard-Jones parameters in physical units. For the S-CH_3 and the S-CH_2 parameters, the ϵ_S is 126 K and the σ_S is 3.55Å. The unlike site interaction parameters are determined using the geometric mean of the like site parameters. For example, σ for the CH_3-S interaction is $[3.55 \times 3.905]^{\frac{1}{2}} = 3.723$Å.

Sites	σ, Å	ϵ, K
S_I-S_I	4.25	200.
CH_3-CH_3	3.905	88.1
CH_2-CH_2	3.905	59.4
CH_3-CH_2	3.905	72.3
CH_3-S	3.723	105.4
CH_2-S	3.723	86.5

Model parameters, program units

The conversion between physical units and program units is based on specific, but arbitrary units for energy, length and mass. The energy in the program is in units of 78.24 K, lengths are in units of 3.16 Å, and masses are in units of 18 unified atomic mass units. With these units, the time unit is τ =1.67 ps. These particular values were chosen to make these quantites consistent with existing NIST codes,[6] particularly with codes that simulate water using the SPC/E model.[7] The following tables contain the model parameters in program units.

Table 6. Stretch parameters in program units. The CH_3 and CH_2 sites are equivalent for the stretch interaction.

Parameter	S-C	C-C
γ_2	57803	57803
d_0	0.576	0.487

Table 7. Bend parameters in program units.

Parameter	C-C-C	S-C-C
γ_θ	798.8	798.8
θ_0, deg	109.5	114.4

Table 8. The coefficients, C_l, in the torsional potential in program units.

l	C_l
0	14.26
1	18.69
2	-20.17
3	-4.70
4	40.34
5	-48.15

Table 9. The parameters for the surface interactions in program units.

Site	C_{12}	C_3	z_0
CH_3	0.4048	8.426	0.2722
CH_2	0.3610	6.927	0.2722
S	0.4393	73.16	0.0851

3. **Discussion of the simulation code**

Operational details

In this part of the discussion, items set in the `typewriter` font refer to source code statements. A FORTRAN77 program, `mdsxx.f`, implements the potential model and generates the resulting dynamics for a fixed number of long chain molecules "bound" to the surface

Table 10. Lennard-Jones parameters in program units. For the S-CH_3 and the S-CH_2 parameters, the ϵ_S is 1.61 and the σ_S is 1.123.

Sites	σ	ϵ
S_I-S_I	1.3450	2.5560
CH_3-CH_3	1.2358	1.1260
CH_2-CH_2	1.2358	0.7592
CH_3-CH_2	1.2358	0.9246
CH_3-S	1.1780	1.3470
CH_2-S	1.1780	1.0160

by the $V_{surface}(z)$ interaction. The number of chains and the number of molecules in a chain are "hard-wired" into the code through a *parameter* statement that appears in the main routine and in most subroutines,

```
parameter (nsite=8, nmols=225, np=nsite*nmols),
```

where `nsite` is the number of sites on a chain and `nmols` is the number of chains. The parameter `np` is the total number of atoms in the system. At the start of a run, the program reads two files, `mdx2` and `mdx8` that are named in *open* statements.

```
open(2,file='mdx2')
open(3,file='mdxout')
open(8,file='mdx8')
```

The file `mdxout` is the output file to be discussed below. The file `mdx2` contains Lennard-Jones interaction parameters, the surface-site parameters, the size of the periodic simulation cell (x and y dimensions are subject to periodic boundary conditions), the duration of the simulation, the size of a time step, and related quanties. The file `mdx8` contains a set of coordinates and momenta of the atoms that make up the system. This file also contains a molecule number for each chain and the forces acting on each atom. The forces corresponding to the coordinates are needed by the Beeman algorithm that is employed to integrate the equations of motion.[8] The configuration defined by this file is the initial condition used at the start of a run. This file is rewritten periodically during a run and at the end of a run. The parameter `jrest` determines the frequency of the rewrites.

The operation of the code requires two preparatory steps. The first step is to edit the first line of `mdx8` so that the job begins at time t=0. Here is a example of how this line changes. Before editing the first line might read

```
10000        1800 -0.230036E+02  0.569967E+01
```

and after editing the line would read

```
    0000       1800 -0.230036E+02  0.569967E+01
```

The other quantites in this line are `npart`, the number of atoms in the system (not the number of chains), `etot`, the total energy-per-atom and `ek`, the kinetic energy-per-atom. Normally, these would not be modified.

The second step is to edit `mdx2` so that the time step, the number of time steps, and the energy have the desired values. An example of `mdx2` is shown here with all quantities listed in *program* units.

```
       0       10000       1800       1000
    .0006   -23.00      32.0        3.75       3.75       3.75
     .25560+01       .13450+01       .00000+01       .18333+01
     .10160+01       .11780+01       .00000+00       .00000+01
     .13470+01       .11780+01       .00000+00       .00000+01
     .75920+00       .12358+01       .00000+00       .77780+00
     .92460+00       .12358+01       .00000+00       .00000+00
     .11260+01       .12358+01       .00000+00       .83333+00
     .43930+00       .73160+02       .08513-00       .00000+00
     .36100+00       .69270+01       .27220-00       .00000+00
     .40480+00       .84260+01       .27220-00       .00000+00
```

The first line contains `j0, jmax, npart,` and `jrest`. The quantity `j0` is kept at zero, the quantity `jmax` is the number of time steps in the run, the quantity `npart` is the number of atoms in the system and *must equal* `np`, and `jrest` determines the number of time steps between updates of `mdx8`. The second line contains `dt`, the duration of a time step (0.0006 in program units is 1 fs in physical units), `ene`, the energy-per-atom, `xmax`, the length of a side of the periodic simulation cell, and `rm1, rm2, rm3` that define the range of the Lennard-Jones interactions. The other quantites in this file specify the Lennard-Jones parameters, the surface interaction parameters and the masses of the "atoms". The order of the Lennard-Jones and surface interaction parameters is set in the code by this *read* statement.

```
      read(2,3)(epsi(j),sig(j),alf(j),ymx(j),j=1,9)
3     format(4e15.5)
```

The first six values for `epsi` and `sig` are the Lennard-Jones ϵ and σ paramters that are used in the generation of energy and force tables, `ve(i,j)`, `vf(i,j)`, i=1,6. The next three rows of `mdx2` contain the surface interaction parameters, C_{12}, C_3, and z_0 in the first three columns. These parameters are used to generate `ve(i,j)`, `vf(i,j)`, i=7,9 These arrays are used to generate the energies and forces during the run. The model is realized in these arrays. Linear interpolation is used to determine the force and energy when the separation of two sites lies between 0.01j and 0.01(j+1). The site pairs are listed here.

```
c     ve(1,j) SII parameters
c     ve(2,j) S-CH2
```

```
c       ve(3,j) S-CH3
c       ve(4,j) CH2-CH2
c       ve(5,j) CH2-CH3
c       ve(6,j) CH3-CH3
c       ve(7,j) S-surface
c       ve(8,j) CH2-surface
c       ve(9,j) CH3-surface
```

The fourth column of the file contains `ymx`, the masses of the sites.

Both of these files are read using explicit *format* statements so it is important that the positions of the characters NOT change during the editing process. If they do, *unpredictable* and almost certainly *undesirable* things will happen.

If the compiled version of `mdsxx.f` is named `mdsxx`, execution of the program starts when

```
mdsxx
```

is typed on the console.

Further detail on the program structure

Here we examine the code in more detail. Starting with the main program, we follow the sequence of operations that occur as the simulation procedes. The first line of the program defines the main program:

```
      program mdsxx
```

The first set of tasks is to read input information, to set initial values of several quantities to zero, to generate the energy and force arrays, to establish a neighbor table, and to scale the momenta so that the desired value of the energy is realized. The statement

```
        call pot(epsi,sig,alf)
```

results in the generation of the enegy and force arrays. The statements

```
c       set up neighbor table
        isum = 0
        call table(isum)
```

produce the neighbor table used to facilitate the calculation of the intermolecular contributions to the energy and the forces. This subroutine also constructs pair distribution functions. The quantity `isum` counts the number of samples. (The neighbor table is reconstructed every six time steps. If systems with larger numbers of molecules are simulated, it would be advisable to consider alternative methods to construct the table. The cell index method that scales with the number of molecules would be a good candidate.[9]) At this point, the simulation is ready to begin. The

```
100     continue
```

statement marks the return point after each time step and the related processing is completed.

The integration of the equations of motion occurs in the subroutine `driver`.

```
        call driver(ek,etot,zmx,dtv,dt6,den)
```

This subroutine uses the Beeman algorithm[8] to integrate the equations of motion. This is a multiple-time scheme that first updates the postions of the particles according to

$$x(t+\Delta t) = x(t) + v(t)\Delta t + \frac{1}{6}[4a(t) - a(t-\Delta t)]\Delta t^2$$

where $x(t)$, $v(t)$, and $a(t)$ are the x-coordinate, the x-component of the velocity, and the x-component of the acceleration of the particle at time t. The quantity Δt is the time step; `dt` in the code and in `mdx2`. As noted above, $\Delta t = 1$ fs in physical units. This size step leads to good energy conservation, even with the strong harmonic interactions between intramolecular sites. Next, the accelerations for the new positions are calculated with the statement

```
        call accel(axp,ayp,azp,etot)
```

where `axp` is $a(t+\Delta t)$ and `etot` is the energy of the system. The accelerations are obtained in two steps. First the intramolecular terms and the surface terms are obtained with the call

```
            call force2(fx,fy,fz,ct,xmax,xmax2,e2,e3,v2,v3,cphi,j)
```

where `fx` is that part of $a(t+\Delta t)$. This subroutine also evaluates the contribution of these terms to the potential energy as a function of the distance above the surface. Upon completion of the calls to `force2`, the intermolecular terms are evaluated with the aid of the neighbor table as indicated in this code fragment.

```
        nx=1
        do 50 j=1,np-1
c       Begin njk determination
         mj=mod(j,nsite)
          if(mj.gt.1) then
             mj=2
          else if(mj.eq.1) then
             mj=0
          else
             mj=3
          end if
30          k=lr(nx)
           nx=nx+1
          if(k.eq.0) go to 50
```

The array `lr` contains the neighbors of particle j with index k greater than j. Note that a zero is used to indicate the end of the list of neighbors of particle j. The quantity `mj` and the corresponding quantity mk are needed to determine which ve and vf arrays are to be used for the energy and forces.

Upon the return to driver, the velocities of the particles are updated by

$$v(t + \Delta t) = v(t) + \frac{1}{6}[2a(t + \Delta t) + 5a(t) - a(t - \Delta t)]\Delta t.$$

The kinetic energy is calculated completing the integration process. Note that the code uses momenta and forces so these quantites are divided by the mass to produce velocities and accelerations. The time step counter jt is incremented upon the return from driver.

During a simulation run, several quatities are generated and the results saved for later processing. Some examples are included here to illustrate some of these quantities. The following three figures are for a system of 225 molecules with an area per molecule of 30 Å^2. The site number distribution is determined as a function of z, the distance above the surface and is in the array pden. Potential energy profiles for the intramolecular terms, the intermolecular terms, and the surface interaction terms are generated as functions of z in arrays utra, uitr, and uwal respectively. No distinction is made concerning headgroups, surface sites, and intermediate sites. The site density and various energy densities are displayed in Figure 3. These densities are constructed with a resolution of 0.01σ in z so that schematically,

$$n(z) = \mathtt{pden}(z)/(0.01 * \mathtt{xmax} * *2),$$

with similar expressions for the energy densities. The energy density is assumed to be localized on site positions.

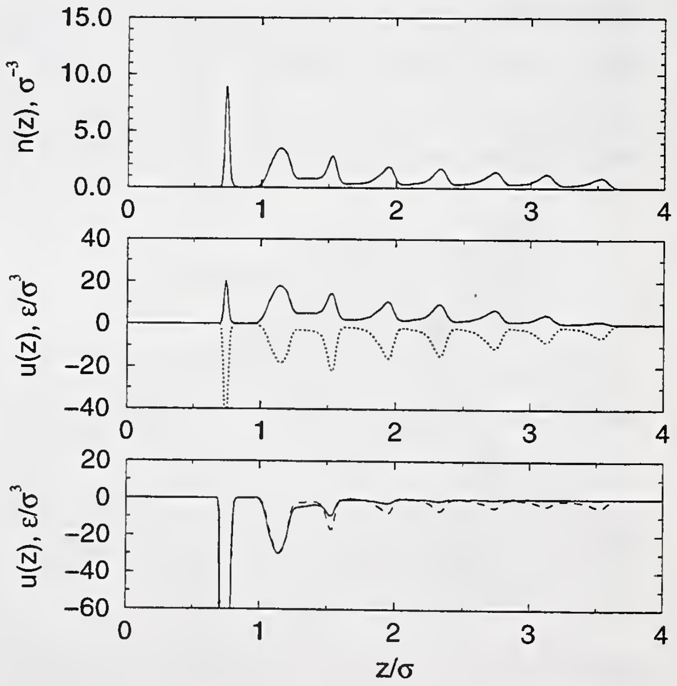


Figure 3. The top plot shows the site density profile as a function of z, the distance above the surface. The middle plot shows the intramolecular energy density (solid line) and the intermolecular energy density (dotted line). The bottom plot shows the surface energy density (solid line) and the total potential energy density (dashed line).

Several time correlation functions are constructed over a 5 ps time interval by the subroutine dr. First, a velocity-velocity time autocorrelation function for all sites is generated in array p. Also, several mean-square displacement functions are constructed. These displacements are for the surface sites, array pu, the end sites, array pc, and for all sites, array rrr. Time averaged values for the kinetic energy and the total energy are listed in 2000Δt blocks.

The distribution for the cosine of the bend angle is in ct, and the distribution of the cosine of the twist angle is in array cphi. These distributions are averaged over all of the chains and placed in the array g(2,j) in locations 1-100 for the bend angle and in locations 101-201 for the torsion angle. The cosines are sorted with a resolution of 0.02. An angle describing the orientation of the chains relative to the surface is determined. The cosine of χ, the angle made by the line joining the thiol site near the surface to the CH_3 end group site and the normal to the surface, is generated and the distribution of these cosines is in the array pang. These distributions are normalized so that

$$\int_{-1}^{1} dx P(x) = 1.$$

These distributions are displayed in Figure 4.

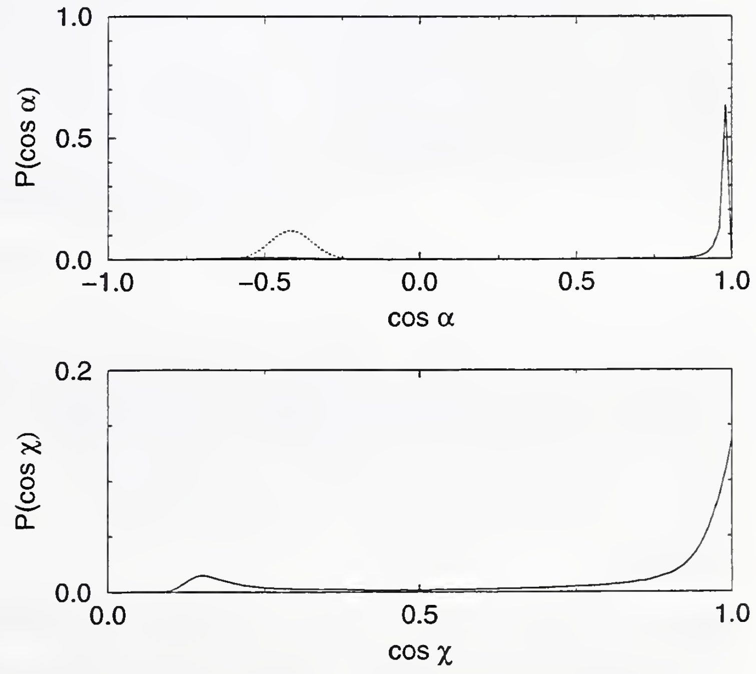


Figure 4. In the upper plot, the distributions for the bend angles (dashed line) and for the torsion angles (solid line) are indicated. The lower plot is the distribution of the angles the molecules make relative to the normal to the surface.

The pair distribution functions for intramolecular sites and all sites are constructed in the subroutine table and are stored in the arrays g(1,j) and g(3,j) with a resolution of

0.01 program unit. Figure 5 is an example of these distributions with the factor $4\pi r^2 dr$ removed. The density factor that is usually removed in liquid state studies has not been removed here. Hence we use the notation $\rho(r)$ rather than g(r). That is, ρ(r) is related to the probability of finding an atom at a given distance r from an atom. As shown in Figure 5, the first and second intramolecular neighbor features in ρ(r) are quite sharp with a large amplitude. The differences between the intramolecular and intermolecular distributions are quite pronounced and are a result of strong constraints on a molecule's structure. An eight site molecule is only three σ units long so the larger separations are dominated by intermolecular correlations. This distribution function provides a rather different "view" of the molecular environment than does the density profile, n(z).

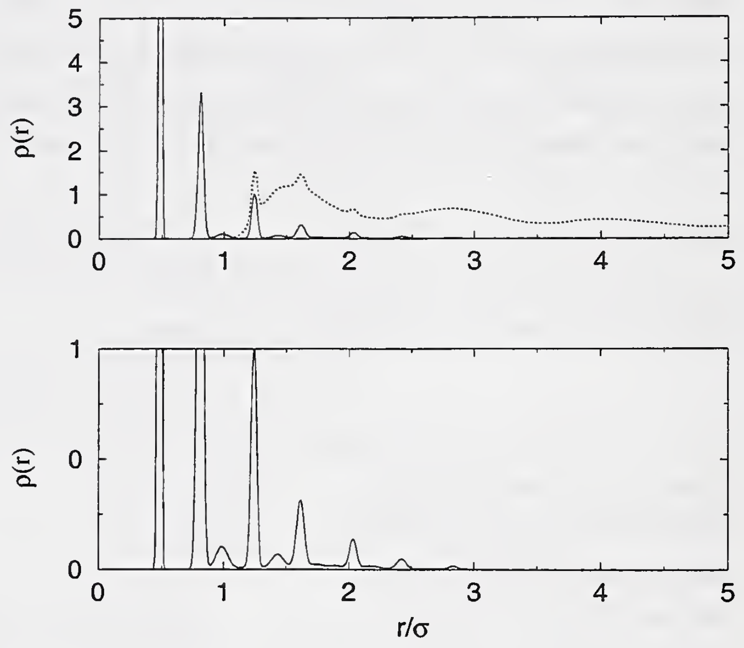


Figure 5. The lower plot contains the intramolecular distribution only while both the intramolecular and total distributions are shown in the upper plot as solid and dashed lines respectively.

If the time step counter, `jt` is less than the quantity `jmax` when these analysis steps are completed, the program is directed back to the

```
100     continue
```

statement.

When `jt` equals `jmax`, the various quantities are normalized, and written to file `mdxout`. The configuration at the end of the run is written to file `mdx8` and the run stops.

How can the temperature of the system be changed?

It might seem that changing the value of `ek` in `mdx8` would work, but that is not the case. A provision for changing the momenta of all particles is provided at the beginning of a run.

This is done by comparing the quantities ene and etot and then adjusting the momenta of the particles as described in this fragment of the code.

```
c      scale to desired energy and assign masses.
       scale=1.+(ene-etot)/ek
       if(scale.gt.0.)  then
        scale=sqrt(scale)
       else
        scale=0.
       end if
        write(6,19) ene,etot,ek,scale
19      format(1x,4e13.4)
        do 22 j=1,np
         mj=mod(j,nsite)
         if(mj.eq.1) then
           zmx(j)=1./ymx(1)
         else if(mj.eq.0) then
           zmx(j)=1./ymx(6)
         else
           zmx(j)=1./ymx(4)
         end if
          u(j)=u(j)*scale
          v(j)=v(j)*scale
          w(j)=w(j)*scale
22      continue
```

The do 22 loop also assigns the proper 1/mass parameters to each of the particles. This energy rescaling procedure requires some practice to achieve a particular final temperature as the balance between kinetic and potential energy is disturbed. Usually a reasonable balance is achieved within a few ps (thousands of time steps). A rough estimate of the temperature change is provided by assuming that about one-half of the energy change will go into potential energy and the rest will go into kinetic energy.

How can the area of the system be changed?

Changing the area involves two steps. The first step is easy, just change the value of xmax in line 2 of the input file mdx2. The second step is to rescale the x- and y-components of all of the atoms by the ratio of the new value of xmax to the old value. This step is *essential* as the periodic boundary conditions can have part of a molecule at one edge of the cell and the rest of it at the opposite edge. Unless every thing is scaled properly, the change in xmax produces large distortions of the stretch and bend degrees of freedom of molecules that are at the edges of the simulation cell.

4. **Sample results**

The starting configuration for these simulations was a 21σ x 21σ square array of eight-site molecules (heptane-thiol) attached to the surface in a square lattice arrangement. A total of 225 (15^2) chains standing normal to the surface provided the starting configuration.

During equilibration, the energy was adjusted so that the temperature was approximately room temperature. Once the internal degrees of freedom settled down, it was found that the molecules had migrated into a hexagonal arrangement with one defect.

A series of runs at increasing area per molecule were made, keeping the temperature near room temperature. As the area increased, more of the chains assumed a configuration nearly parallel to the surface, although some regions with nearly vertical chains persisted. This and other features of this tethered chain system are illustrated in the following set of figures.

First, we examine the relatively high density system with an area of 21.4 Å^2 per molecule. This density is what is found for self-assembled alkane-thiol films on the 111 surface of gold in the dense phase. As shown in Figures 6 through 8, the molecules are primarily in an upright configuration with very few gauche defects. The overall potential energy is negative indicating that the cohesion of the chains is energetically favored. The density profile is rather sharply structured, a feature in keeping with the distribution of $\cos\chi$.

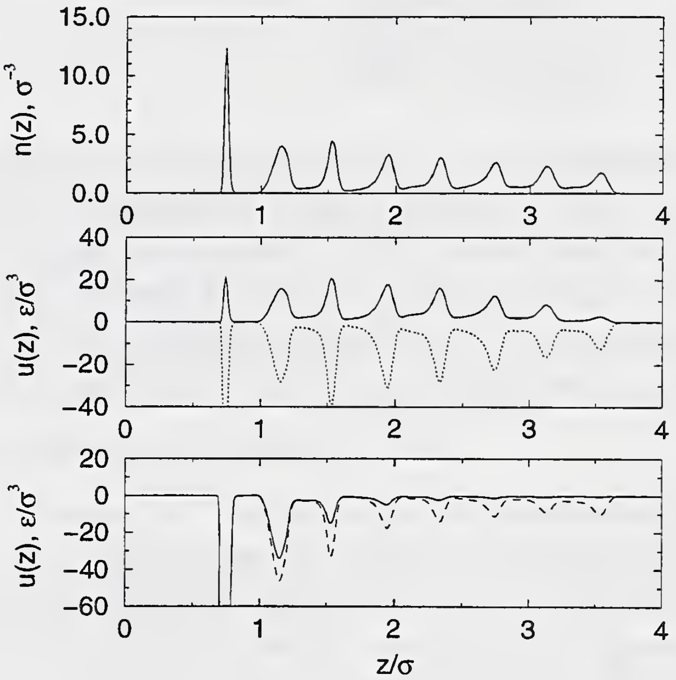


Figure 6. The density and energy profiles for the 21.4 Å^2 state.

A “snapshot” of the configuration of the molecules indicates that this system is ordered.

Next, we consider an expanded state with an area of 25.6 Å^2 per molecule.

The distribution of $\cos\chi$ for this state indicates that an increasing number of chains are lying down. Even so, the total potential energy remains negative and the density profile remains quite structured suggesting that the system retains most of the order present in

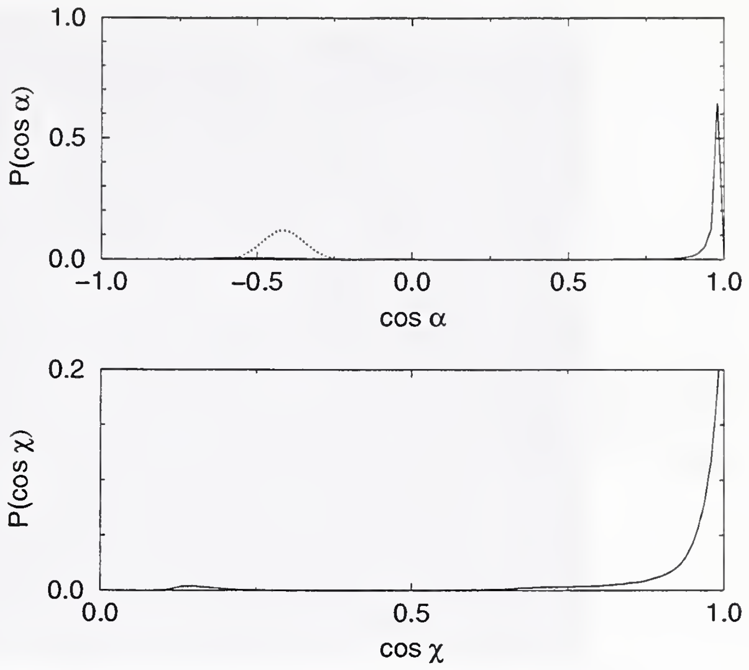


Figure 7. The conformation and orientation distributions for the 21.4 Å^2 state.

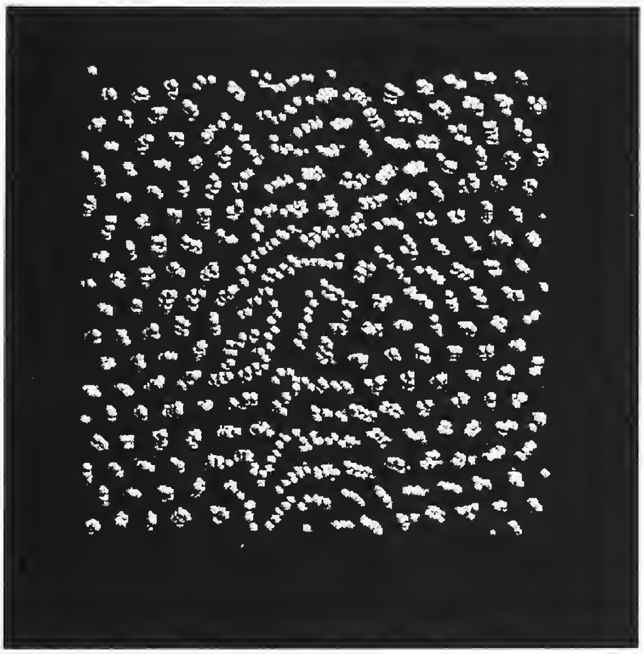


Figure 8. A conformation of the 21.4 Å^2 state.

the higher density state. A "snapshot" of the system indicates that the disordered region is confined to a small region of the surface as an extended defect or domain boundary region.

Finally, we examine a much lower density state with an area of 45.4 Å^2 per molecule. For this state, the density profile is much more diffuse and the energy density is dominated by surface energy terms. There is a considerable increase in the number of molecules

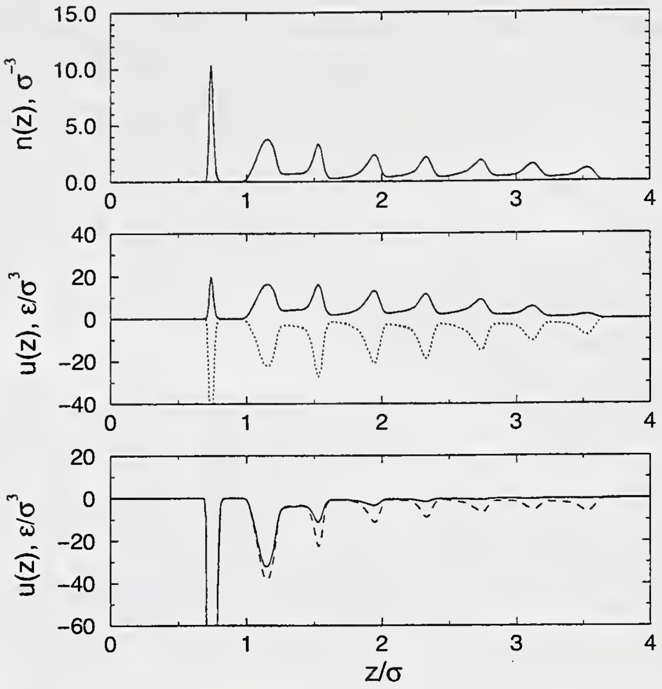


Figure 9. The density and energy profiles for the 25.6 Å^2 state.

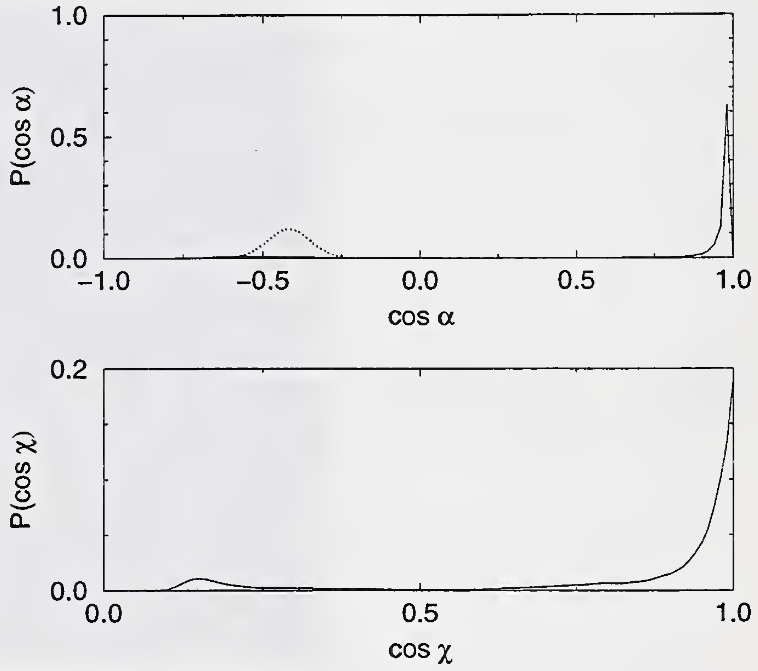


Figure 10. The conformation and orientation distributions for the 25.6 Å^2 state.

lying down. A "snapshot" indicates that there is an ordered domain coexisting with a low density region of molecules with a broad distribution of orientations relative to the surface.

Finally, we show how the energy varies with the area/molecule along the T = 285 K isotherm for the 8 site system. Note that the change in energy with increasing area is quite slow for the higher area/molecule states. This indicates that the change in the order

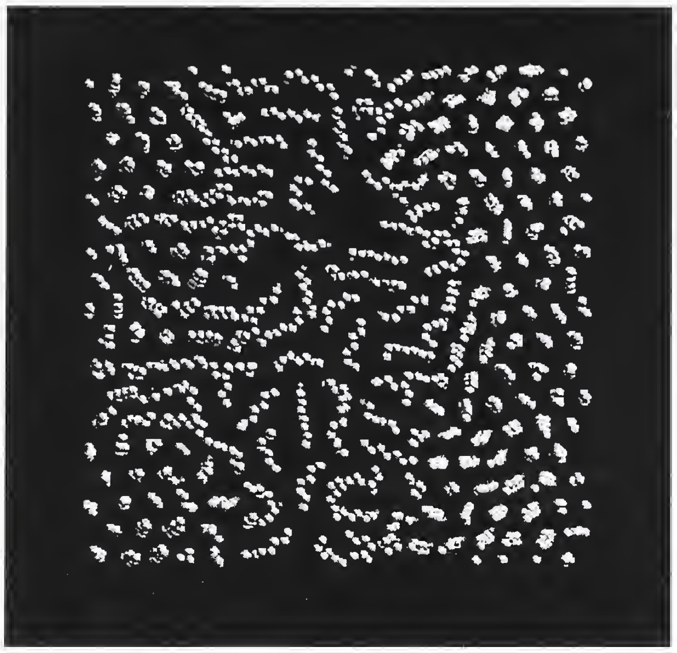


Figure 11. A conformation of the 25.6 Å^2 state.

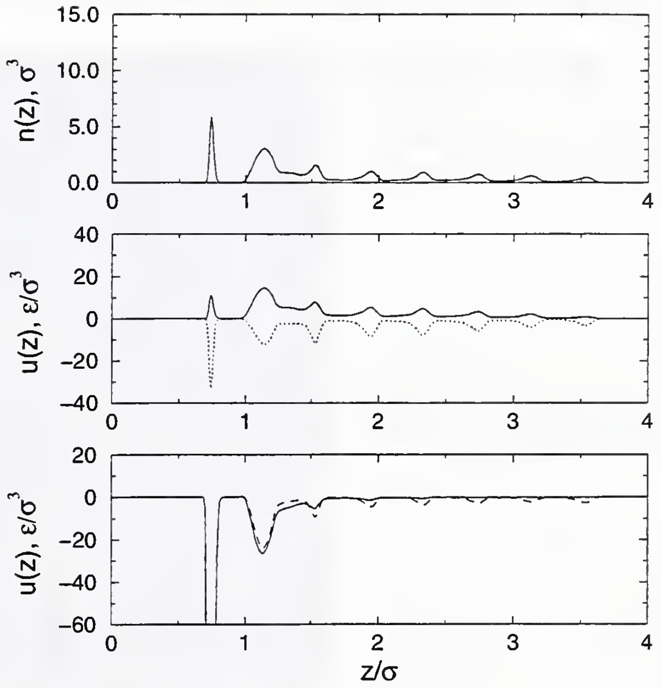


Figure 12. The density and energy profiles for the 45.4 Å^2 state.

of the system occurs gradually once a fraction of the molecules are no longer in the upright configuration.

5. How to modify features of the simulation

The code `mdsxx.f` contains a number of features that might be changed to "improve" the model. Some changes will require modification of the code followed by recompilation and

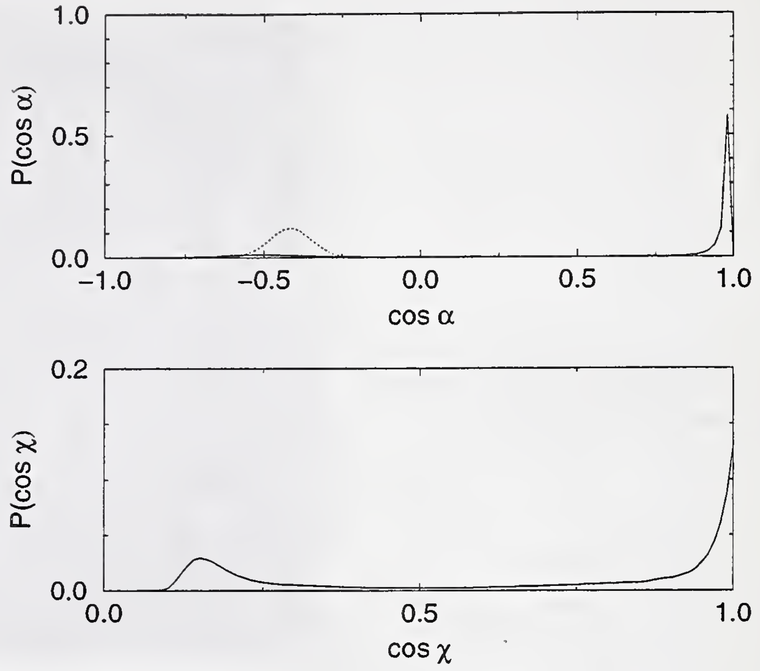


Figure 13. The conformation and orientation distributions for the 45.4 Å^2 state.

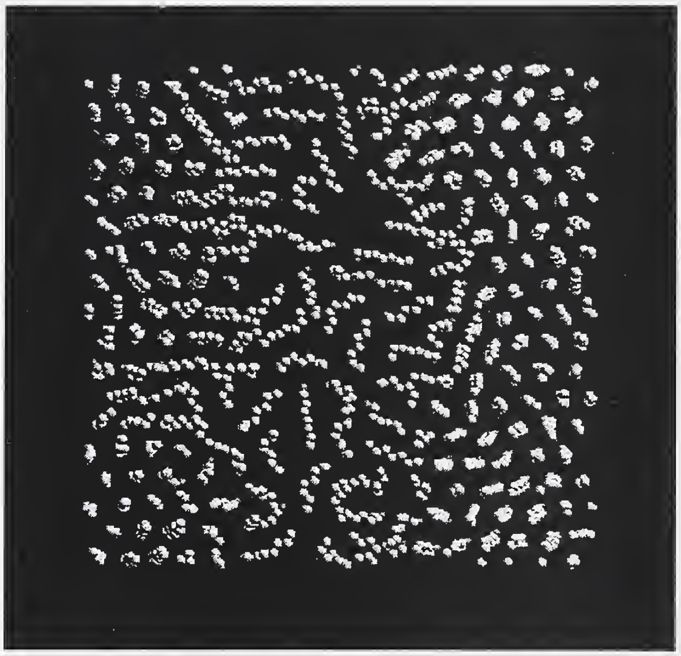


Figure 14. A conformation of the 45.4 Å^2 state.

testing, while other changes can be implemented by changing numbers in the file mdx2. In this section we summarize what is needed to make various changes in the model.

First, we list modifications that do not require changes in the code. As noted at the end of Section 3, the area of the system can be changed by adjusting xmax in mdx2 and by rescaling the x- and y-coordinates in mdx8. The Lennard-Jones potential parameters and the z-dependent surface interaction parameters are in the file mdx2 so they can be readily

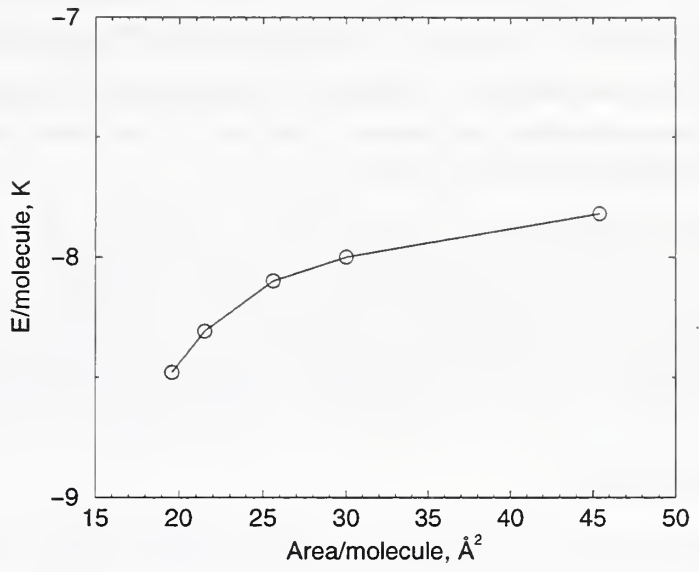


Figure 15. The energy/molecule (in K) is shown as a function of the area/molecule (in Å^2) for the T = 285 K isotherm.

changed. The current values of these parameters are listen in Tables 9 and 10.

Other changes will require modifications of the source code. Changing the number of chain molecules and/or the number of sites per chain can be accomplished in three steps. The first step is to modify all instances of the parameter statement

```
        parameter (nsite=8, nmols=225, np=nsite*nmols)
```

and then recompile the code. The second step is to edit the files `mdx2` and `mdx8` so that `npart` matches `np`. The third step is to generate a new file `mdx8` with the correct number of entries and then to "equilibrate" the system. If the number of chains is increased significantly, it may be desirable to restructure the subroutine `table`.[9]

The simulation cell is a square with a side of length `xmax`. It would be possible to change this to a rectangle by introducing a second quantity *ymax*. It would also be necessary to modify the main routine and several subroutines where periodic boundary conditions are imposed. Searching for the strings `.gt.xmax`, `.gt.xmax2`, `.gt.XMAX`, and `.gt.XMAX2` will locate *most*, but not necessarily *all* places where changes are required.

At present, the intramolecular interaction parameters for the stretch, bend, and torsion interactions of the chain are part of the subroutine `force2`. The current values of these parameters are listed in Tables 6 through 8. Changes in these parameters, or in the form of the interactions will involve code modification and recompilation. One possibility would be to modify the code so that the parameters are read from `mdx2`, reducing the number of recompilations needed.

The surface interaction is a function of the distance, z, above the surface. Inclusion of x- and y-dependence of the surface interaction would require modifications to `force2`.[5]

The intermolecular Lennard-Jones interaction is currently the same as the intramolecular Lennard-Jones interaction. Changes in the form of either interaction would entail code modification of subroutine `pot`.

The molecules are considered to be linear chains with the thiol, methylene, and methyl groups treated as united atoms. Any modification of the structure of the molecules would require considerable recoding since the linear sequence of intramolecular sites is incorporated into the `force2` subroutine and other places as well. Currently these are structureless atoms so torsion motions of the groups about the chain are excluded.

References

[1] J. Hautman and M. L. Klein, J. Chem. Phys. **91**, 4994 (1989).

[2] B. Smit, S. Karaborni, and J. I. Siepmann, J. Chem. Phys. **102**, 2126 (1995).

[3] S. Toxvaerd, J. Chem. Phys. **93**, 4290 (1990).

[4] J. H. R. Clarke and D. Brown, Mol. Phys. **58**, 815 (1986).

[5] J. Hautman, J. P. Bareman, W. Mar, and M. L. Klein, J. Chem. Soc. Faraday Trans. **87**, 2031 (1991).

[6] R. D. Mountain, NISTIR 6028, National Institute of Standards and Technology (unpublished).

[7] H. J. C. Berendsen, J. R. Grigera, and T. P. Straatsma, J. Phys. Chem. **91**, 6269 (1987).

[8] P. Schofield, Comput. Phys. Commun. **5**, 17 (1976).

[9] R. D. Mountain, NISTIR 5545, National Institute of Standards and Technology (unpublished).

[10] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids* (Clarendon Press, Oxford, 1987), pp. 156–159.

[11] P. M. Morse and H. Feshbach, *Methods of Theretical Physics* (McGraw-Hill, New York, 1953), p. 114.

Appendix 1 Intramolecular interactions.

The intramolecular interaction contributions to the forces and the stresses are examined here.[10] There are four intramolecular interaction terms, the stretch that involves adjacent sites, the bend that involves three sequential sites, the torsion that involves four sequential sites, and the Lennard-Jones interaction between sites separated by three or more intervening sites. As we are concerned with linear chain molecules, it is convenient to label sites in a molecule sequentially. Thus, the stretch interaction involves sites *j* and *j+1*, the bend interaction involves sites *j, j+1,* and *j+2*, and the torsion interaction involves sites *j* through *j+3*. We use the following notation to indicate the relative positions of the sites. Site *j* is at position $\mathbf{R}_j$ and site *j+1* is at position $\mathbf{R}_{j+1}$. The relative position vector is

$$\mathbf{r}_j = \mathbf{R}_{j+1} - \mathbf{R}_j.$$

This vector is directed toward site *j+1*. The magnitude of $\mathbf{r}_j$ is r_j and x_j is the x-component of the vector $\mathbf{r}_j$. This notation *matches* the symbols used in the code. Now let us examine the individual terms in the intramolecular interaction and the resulting contributions to the force.

The harmonic stretch energy, for a pair of sites *j, j+1* is $\frac{1}{2}\gamma(\mathrm{r}_j\text{-}\mathrm{d}_0)^2$ and the x-component of the force on site *j+1* is

$$F^x_{j+1} = -\gamma(r_j - d_0)x_j/r_j$$

and $F^x_j = -F^x_{j+1}$.

The harmonic bend energy, for the triple of sites *j, j+1,* and *j+2* is

$$V_3 = \frac{1}{2}\gamma_3(\cos\theta - \cos\theta_0)^2$$

where θ is the angle with site *j+1* as the vertex and

$$\cos\theta = -\frac{\mathbf{r}_j \cdot \mathbf{r}_{j+1}}{r_j r_{j+1}}.$$

The energy can be expressed in terms of the bond between sites *j, j+1* and of the bond between sites *j+1, j+2* so that the forces on the sites can also be represented as contributions from the bonds. This will make the expressions for the forces easier to organize.

Since site *j* enters the three body energy only through $\mathbf{r}_j$, the force on site *j* is

$$F^x_j = -\frac{\partial V_3}{\partial x_j}\frac{\partial x_j}{\partial X_j} = a$$

and the force on site *j+2* is

$$F^x_{j+2} = -\frac{\partial V_3}{\partial x_{j+1}}\frac{\partial x_{j+1}}{\partial X_{j+2}} = b.$$

Here, X_j is the x-component of $\mathbf{R}_j$. The force on site $j+1$ is

$$F^x_{j+1} = -\frac{\partial V_3}{\partial x_j}\frac{\partial x_j}{\partial x_{j+1}} - \frac{\partial V_3}{\partial x_{j+1}}\frac{\partial x_{j+1}}{\partial X_{j+1}} = -a - b.$$

Since

$$\frac{\partial x_{j+1}}{\partial X_{j+2}} = 1$$

and

$$\frac{\partial x_j}{\partial X_j} = -1,$$

this illustrates how the force can be decomposed into "bond" contributions. The term a is from the *j, j+1* bond and the term b is from the *j+1, j+2* bond. Since these are intramolecular forces, the sum of the forces are zero.

A similar analysis is possible for the torsion, four-body terms. The first step is to express $\cos(\phi)$ in terms of the bonds. The dihedral angle ϕ is the angle between the normals to the planes determined $\mathbf{r}_j$ and $\mathbf{r}_{j+1}$ and by $\mathbf{r}_{j+1}$ and $\mathbf{r}_{j+2}$. Let $\mathbf{m}_1$ be the normal to the first plane, namely the vector product

$$\mathbf{m}_1 = \mathbf{r}_j \times \mathbf{r}_{j+1}.$$

The normal to the second plane $\mathbf{m}_2$ is

$$\mathbf{m}_2 = \mathbf{r}_{j+1} \times \mathbf{r}_{j+2}.$$

The dihedral angle is then determined using

$$\cos(\phi) = -\frac{\mathbf{m}_1 \cdot \mathbf{m}_2}{\mid \mathbf{m}_1 \mid\mid \mathbf{m}_2 \mid}.$$

This leads to the following expression when the vector identity [11]

$$(\mathbf{A} \times \mathbf{B}) \cdot (\mathbf{C} \times \mathbf{D}) = (\mathbf{A} \cdot \mathbf{C})(\mathbf{B} \cdot \mathbf{D}) - (\mathbf{A} \cdot \mathbf{D})(\mathbf{B} \cdot \mathbf{C})$$

is used to reduce the vector products to scalar products;

$$\cos(\phi) = -\frac{\left[(\mathbf{r}_j \cdot \mathbf{r}_{j+1})(\mathbf{r}_{j+1} \cdot \mathbf{r}_{j+2} - (\mathbf{r}_j \cdot \mathbf{r}_{j+2})r^2_{j+1}\right]}{\left[r^2_j r^2_{j+1} - (\mathbf{r}_j \cdot \mathbf{r}_{j+1})^2\right]^{\frac{1}{2}}\left[r^2_{j+1} r^2_{j+2} - (\mathbf{r}_{j+1} \cdot \mathbf{r}_{j+2})^2\right]^{\frac{1}{2}}}.$$

Now it is a straightforward task to generate the forces in terms of the bond contributions.

Appendix 2. The source code.

```
      program mdsxx
c     Compile on R8000 using f77 -64 -mips4 -O3
c       RDM  9-12-97.    Last modification 2-23-98
c     MD code for long chains on a surface.    Revised 9-17-96
c     ve(1,j) SII parameters
c     ve(2,j) S-CH2
c     ve(3,j) S-CH3
c     ve(4,j) CH2-CH2
c     ve(5.j) CH2-CH3
c     ve(6,j) CH3-CH3
c     ve(7,j) S-surface
c     ve(8,j) CH2-surface
c     ve(9,j) CH3-surface
c     g is used for atom-atom, cm-cm, bond angles.
      parameter (nsite=8, nmols=225, np=nsite*nmols)
      dimension x(np),y(np),z(np),u(np),v(np),w(np)
      dimension ax(np),ay(np),az(np),axm(np),aym(np),azm(np)
      dimension nq(np),lr(300000)
      dimension ve(9,800),vf(9,800),die(9,800),dif(9,800)
      dimension epsi(9),sig(9),alf(9),ymx(9),zmx(np)
      double precision g(3,500),pden(0:1000),pang(-1:100)
      dimension uu(np),vv(np),ww(np),bx(np),by(np),bz(np),u1(np),
     x   u2(np),u3(np),x0(np),y0(np),z0(np),u0(np),v0(np),w0(np),
     x   pc(4999),ua(np),va(np),wa(np)
     double precision sxx(0:1000),syy(0:1000),szz(0:1000)
     double precision utra(0:1000),uitr(0:1000),uwal(0:1000)
     common /cuuu/ utra,uitr,uwal
     common /cstr/ sxx,syy,szz
c
c     common blocks
c
      common /cord/ x,y,z,u,v,w,nq
      common /cacc/ ax,ay,az,axm,aym,azm
      common /ctab/ lr,g
      common /cvef/ ve,vf,die,dif
      common /cpar/ xmax,xmax2,rm1,rm2,rm3,rp1,rp2,rp3,dt
      common /cgdv/ rrr(4999),p(4999),px,pu(4999),up,pc,p0
      COMMON /CREF/ UU,VV,WW,bX,bY,bZ,u1,u2,u3,x0,y0,z0,u0,v0,w0,
     x     ua,va,wa
c
c     open files
c
      open(2,file='mdx2')
      open(3,file='mdxout')
      open(8,file='mdx8')
c
c     read in startup information
c
      read(2,1) j0,jmax,npart,jrest
1     format(4i10)
      read(2,2) dt,ene,xmax,rm1,rm2,rm3
c
c     rm1,rm2,rm3 are the range of the interactions
c     rp1,rp2,rp3 and the square of the corresponding distances
```

```
c         for use in construction of the neiqhbor table.
2       format(6f10.4)
        read(2,3)(epsi(j),sig(j),alf(j),ymx(j),j=1,9)
3       format(4e15.5)
c
c       list what has been provided
        write(3,1) j0,jmax,npart,jrest
        write(3,2) dt,ene,xmax,rm1,rm2,rm3
        write(3,3)(epsi(j),sig(j),alf(j),ymx(j),j=1,9)
c
c       rework the range parameters
        rp1=(rm1+0.3)**2
        rp2=(rm2+0.3)**2
        rp3=(rm3+0.3)**2
            rm1=rm1**2
            rm2=rm2**2
            rm3=rm3**2
c
c       specify half-box size
        xmax2=0.5*xmax
c
c       generate  pi
          pi=4.*atan(1.)
          dt6=dt*dt/6.
          dtv=dt/6.
c       read in restart information
        call restart(etot,ek,jt,npart,1)
c
c       set up potential and force
        call pot(epsi,sig,alf)
c
c       scale to desired energy and assign masses.
        scale=1.+(ene-etot)/ek
        if(scale.gt.0.)  then
         scale=sqrt(scale)
        else
         scale=0.
        end if
         write(6,19) ene,etot,ek,scale
19       format(1x,4e13.4)
         do 22 j=1,np
          mj=mod(j,nsite)
          if(mj.eq.1) then
            zmx(j)=1./ymx(1)
          else if(mj.eq.0) then
            zmx(j)=1./ymx(6)
          else
            zmx(j)=1./ymx(4)
          end if
           u(j)=u(j)*scale
           v(j)=v(j)*scale
           w(j)=w(j)*scale
22       continue
c
c       set initial values
           ek1=0.
```

```
            ek2=0.
            et1=0.
            et2=0.
            tek1=0.
            tek2=0.
            tet1=0.
            tet2=0.
c      Quantities beginning with t are for block averages.
       do 10 k=1,3
        do 9 j=1,500
         g(k,j)=0.d0
9       continue
10     continue
         px=0.
         up=0.
         p0=0.
         pa=0.
         xc=0.
       do 11 j=1,4999
         rrr(j)=0.
         p(j)=0.
         pu(j)=0.
         pc(j)=0.
11     continue
       do 13 j=0,1000
         pden(j)=0.d0
         sxx(j)=0.d0
         syy(j)=0.d0
         szz(j)=0.d0
         utra(j)=0.d0
         uitr(j)=0.d0
         uwal(j)=0.d0
13     continue
       do 15 j=-1,100
         pang(j)=0.d0
15     continue
c      set up neighbor table
       isum = 0
       call table(isum)
100    continue
c       write(6,101)
c101    format(' call driver')
       call driver(ek,etot,zmx,dtv,dt6,den)
         jt=jt+1
c          write(6,102) jt
c102       format(' jt =',i5)
            ek1=ek1+ek
            ek2=ek2+ek*ek
            et1=et1+etot
            et2=et2+etot*etot
            tek1=tek1+ek
            tek2=tek2+ek*ek
            tet1=tet1+etot
            tet2=tet2+etot*etot
c      Density profile
          do 113 j=1,np
            ix=100*z(j)
```

```
             pden(ix)=pden(ix)+1.
113        continue
c       Surface orientation
             nm=nsite-1
           do 115 j=1,np,nsite
             dz=z(j+nm)-z(j)
             dy=y(j+nm)-y(j)
               if(dy.gt.xmax2) then
                  dy=dy-xmax
               else if(dy.lt.-xmax2) then
                  dy=dy+xmax
               end if
             dx=x(j+nm)-x(j)
               if(dx.gt.xmax2) then
                  dx=dx-xmax
               else if(dx.lt.-xmax2) then
                  dx=dx+xmax
               end if
             rr=sqrt(dx*dx+dy*dy+dz*dz)
             theta=dz/rr
             ix=100*theta+1
             if(ix.lt.0) then
                ix=-1
             else if(ix.gt.100)  then
                ix=100
             end if
             pang(ix)=pang(ix)+1.d0
115        continue
c       Two-ps block averages--hence 2000
           if(mod(jt,2000).eq.0) then
             tek1=tek1/2000.
             tek2=tek2/2000.-tek1*tek1
             tet1=tet1/2000.
             tet2=tet2/2000.-tet1*tet1
            write(3,31) tek1,tet1
            write(3,31) tek2,tet2
             tek1=0.
             tek2=0.
             tet1=0.
             tet2=0.
           end if
          call dr(x,y,z,u,v,w,xmax,npart,jt)
          if(mod(jt,6).eq.0) call table(isum)
          if(mod(jt,jrest).eq.0) call restart(etot,ek,jt,npart,2)
          write(6,71) jt, ek,etot
71        format(i6,3e15.4)
          if(jt.lt.jmax) go to 100
c
c      prepare output
          qq=1./(1.*jt)
           ek1=ek1*qq
           ek2=ek2*qq-ek1*ek1
           et1=et1*qq
           et2=et2*qq-et1*et1
          write(3,31) ek1,et1
          write(3,31) ek2,et2
          write(6,31) ek1,et1
```

```
         write(6,31) ek2,et2
31       format(4e15.6)
        write(3,31) px,0.,0.,0.
        write(3,31)(p(j),pu(j),pc(j),rrr(j),j=1,4999)
c
c       process pair function results
        do 300 j=1,500
          r=0.01*j
          vol=4.*pi*(r*r+.01*r+.01*.01/3.)*.01
          vol=vol/2.
             g(1,j)=g(1,j)/(isum*npart*vol)
             g(2,j)=g(2,j)/(1.*jmax)
             g(3,j)=g(3,j)/(isum*npart*vol)
300     continue
        write(3,33)(j,g(1,j),g(2,j),g(3,j),j=1,500)
33      format(i5,3e15.4)
c
c       density profile
          do 213 j=0,1000
             pden(j)=pden(j)/(1.*jmax)
             sxx(j)=sxx(j)/(1.*jmax)
             syy(j)=syy(j)/(1.*jmax)
             szz(j)=szz(j)/(1.*jmax)
             utra(j)=utra(j)/(1.*jmax)
             uitr(j)=uitr(j)/(1.*jmax)
             uwal(j)=uwal(j)/(1.*jmax)
213       continue
         write(3,333)(0.01*j,pden(j),sxx(j),syy(j),szz(j),j=0,1000)
         write(3,333)(0.01*j,pden(j),utra(j),uitr(j),uwal(j),j=0,1000)
333      format(5e15.5)
c       Orientation distribution
          do 215 j=-1,100
            pang(j)=pang(j)/(1.*jmax)
215       continue
         write(3,334)(0.01*j,pang(j),j=-1,100)
334      format(2e15.5)
c       save final configuration
        call restart(etot,ek,jt,npart,2)
          write(6,7891) g(2,499)
7891      format(' Integrity check',f10.4)
        close(2)
        close(3)
        close(8)
        stop
        end
        subroutine driver(ek,etot,zmx,dtv,dt6,den)
        parameter (nsite=8, nmols=225, np=nsite*nmols)
        dimension x(np),y(np),z(np),u(np),v(np),w(np)
        dimension ax(np),ay(np),az(np),axm(np),aym(np),azm(np)
        dimension axp(np),ayp(np),azp(np),nq(np),zmx(np)
c       common blocks
c
        common /cord/ x,y,z,u,v,w,nq
        common /cacc/ ax,ay,az,axm,aym,azm
        common /cpar/ xmax,xmax2,rm1,rm2,rm3,rp1,rp2,rp3,dt
c
```

```
c       Use the Beeman algorithm
        do 100 j=1,np
           x(j)=x(j)+(dt*u(j)+dt6*(4.*ax(j)-axm(j)))*zmx(j)
           y(j)=y(j)+(dt*v(j)+dt6*(4.*ay(j)-aym(j)))*zmx(j)
           z(j)=z(j)+(dt*w(j)+dt6*(4.*az(j)-azm(j)))*zmx(j)
c
c        invoke periodic boundary conditions
           if(x(j).gt.xmax) x(j)=x(j)-xmax
           if(x(j).lt.0.0)  x(j)=x(j)+xmax
           if(y(j).gt.xmax) y(j)=y(j)-xmax
           if(y(j).lt.0.0)  y(j)=y(j)+xmax
100     continue
c        write(6,101)
c101     format(' call accel')
        call accel(axp,ayp,azp,etot)
        ek=0.
        eck=0.
c23456789012345678901234567890123456789012345678901234567890123456789012
        do 200 j=1,np
           u(j)=u(j)+dtv*(2.*axp(j)+5.*ax(j)-axm(j))
           v(j)=v(j)+dtv*(2.*ayp(j)+5.*ay(j)-aym(j))
           w(j)=w(j)+dtv*(2.*azp(j)+5.*az(j)-azm(j))
           ek=ek+(u(j)*u(j)+v(j)*v(j)+w(j)*w(j))*zmx(j)
c
c       shift forces
           axm(j)=ax(j)
           aym(j)=ay(j)
           azm(j)=az(j)
           ax(j)=axp(j)
           ay(j)=ayp(j)
           az(j)=azp(j)
200     continue
        ek=ek/(2.*np)
        etot=etot+ek
        return
        end
        subroutine restart(etot,ek,jt,na,ix)
        parameter (nsite=8, nmols=225, np=nsite*nmols)
        dimension x(np),y(np),z(np),u(np),v(np),w(np),nq(np)
        dimension ax(np),ay(np),az(np),axm(np),aym(np),azm(np)
c
c       common blocks
c
        common /cord/ x,y,z,u,v,w,nq
        common /cacc/ ax,ay,az,axm,aym,azm
5       format(2i10,2e14.6)
6       format(6f13.6,i4)
7       format(6f13.6)
        jn=8
         if(ix.eq.2) go to 100
          read(jn,5) jt,na,etot,ek
          read(jn,6)(x(j),y(j),z(j),u(j),v(j),w(j),nq(j),j=1,np)
          read(jn,7)(ax(j),ay(j),az(j),axm(j),aym(j),azm(j),j=1,np)
          return
100      rewind(jn)
          write(jn,5) jt,na,etot,ek
        write(jn,6)(x(j),y(j),z(j),u(j),v(j),w(j),nq(j),j=1,np)
        write(jn,6)(ax(j),ay(j),az(j),axm(j),aym(j),azm(j),nq(j),
     c              j=1,np)
       return
       end
      subroutine pot(epsi,ri,alf)
      dimension epsi(9),ri(9),alf(9)
      dimension ve(9,800),vf(9,800),die(9,800),dif(9,800)
      common /cvef/ ve,vf,die,dif
       pi=4.*atan(1.)
       do 50 k=1,6
        do 45 j=1,800
         r=0.01*j
         ve(k,j)=4.*epsi(k)*((ri(k)/r)**12 -(ri(k)/r)**6)
         vf(k,j)=24.*epsi(k)*(2.*(ri(k)/r)**12-(ri(k)/r)**6)/r
45      continue
50     continue
       do 80 k=7,9
       do 70 j=1,800
        r=0.01*j-alf(k)
        ve(k,j)=epsi(k)/(r**12) - ri(k)/(r**3)
        vf(k,j)=(12.*epsi(k)/(r**12)-3.*ri(k)/(r**3))/r
70     continue
80     continue
       do 95 k=1,9
        do 90 j=1,799
           die(k,j)=ve(k,j+1)-ve(k,j)
           dif(k,j)=vf(k,j+1)-vf(k,j)
90      continue
95     continue
      return
      end
      subroutine table(isum)
      parameter (nsite=8, nmols=225, np=nsite*nmols)
      dimension x(np),y(np),z(np),u(np),v(np),w(np),nq(np)
      dimension lr(300000)
      double precision g(3,500)
c     common blocks
c
      common /cord/ x,y,z,u,v,w,nq
      common /ctab/ lr,g
      common /cpar/ xmax,xmax2,rm1,rm2,rm3,rp1,rp2,rp3,dt
        xmax3=xmax2**2
        nflag=0
        isum=isum+1
        njk=1
        nx=1
        mmm=0
      do 50 j=1,np-1
         do 48 k=j+1,np
           xx=abs(x(j)-x(k))
           yy=abs(y(j)-y(k))
           zz=abs(z(j)-z(k))
           if(xx.gt.xmax2) xx=xx-xmax
           if(yy.gt.xmax2) yy=yy-xmax
           rr=xx*xx+yy*yy+zz*zz
```

```
                  if(rr.gt.rp1) go to 45
          if(nq(j).eq.nq(k)) then
c            Collect intramolecular separation data ONLY.
            go to 45
          end if
            lr(nx)=k
            nx=nx+1
c45          if(rr.gt.xmax3) go to 48
45         continue
                r=sqrt(rr)
                ix=100*r+.5
                if(ix.gt.500) go to 48
                g(njk,ix)=g(njk,ix)+1.
47             if(nq(j).eq.nq(k)) then
                    g(3,ix)=g(3,ix)+1.
               end if
48          continue
              lr(nx)=nflag
              nx=nx+1
        if(nx.gt.300000) then
          write(6,51) nx,j
51        format(2i10)
          stop 'lr overflow'
        end if
50    continue
      return
      end
     SUBROUTINE DR(X,Y,Z,U,V,W,XMAX,NPART,JX)
C    VCF AND R**2   4/23/80
      parameter (nsite=8, nmols=225, mol=nsite*nmols)
     DIMENSION X(mol),Y(mol),Z(mol),RRR(4999),
    X  AX(mol),U(mol),V(mol),W(mol),UU(mol),VV(mol),WW(mol),P(4999),
    Y  AY(mol),AZ(mol),A31(mol),A32(mol),A33(mol),PU(4999)
     DIMENSION U1(mol),U2(mol),U3(mol),x0(mol),y0(mol),z0(mol),
    x xc(mol),yc(mol),zc(mol),uc(mol),vc(mol),wc(mol),u0(mol),
    x v0(mol),w0(mol),pc(4999),ua(mol),ud(mol),
    x va(mol),vd(mol),wa(mol),wd(mol)
     COMMON /CREF/ UU,VV,WW,AX,AY,AZ,u1,u2,u3,xc,yc,zc,u0,v0,w0,
    x  ua,va,wa
     COMMON /CGDV/ RRR,P,PX,PU,UP,pc,p0
     XMAX2=.5*XMAX
       msite=nsite-1
     JQ=MOD(JX,5000)
     IF(JQ.EQ.1) GO TO 140
      JQ=JQ-1
      IF(JQ.LE.0) JQ=4999
     DO 10 J=1,NPART
     P(JQ)=P(JQ)+V(J)*VV(J)+U(J)*UU(J)+W(J)*WW(J)
     XX=X(J)-AX(J)
     YY=Y(J)-AY(J)
     ZZ=Z(J)-AZ(J)
     XX=ABS(XX)
     YY=ABS(YY)
     ZZ=ABS(ZZ)
     IF(XX.GT.XMAX2) XX=XX-XMAX
     IF(YY.GT.XMAX2) YY=YY-XMAX
     RRR(JQ)=RRR(JQ)+XX*XX+YY*YY+ZZ*ZZ
     if(mod(j,nsite).eq.1) then
```

```
          pu(jq)=pu(jq)+xx*xx+yy*yy
        else if(mod(j,nsite).eq.0) then
          pc(jq)=pc(jq)+xx*xx+yy*yy+zz*zz
        end if
10      CONTINUE
        GO TO 150
140     DO 145 J=1,NPART
        UU(J)=U(J)
        VV(J)=V(J)
        WW(J)=W(J)
        PX=PX+UU(J)*UU(J)+VV(J)*VV(J)+WW(J)*WW(J)
        AX(J)=X(J)
        AY(J)=Y(J)
        AZ(J)=Z(J)
145     CONTINUE
150     CONTINUE
        RETURN
        END
         subroutine accel(ax,ay,az,etot)
         parameter (nsite=8, nmols=225, np=nsite*nmols, ns=9)
         dimension lr(300000)
         double precision g(3,500)
         dimension x(np),y(np),z(np),u(np),v(np),w(np),ct(np)
         dimension ax(np),ay(np),az(np),nq(np)
         dimension ve(ns,800),vf(ns,800),die(ns,800),dif(ns,800)
         dimensionfx(np),fy(np),fz(np),cphi(np)
        double precision sxx(0:1000),syy(0:1000),szz(0:1000)
        double precision utra(0:1000),uitr(0:1000),uwal(0:1000)
        common /cuuu/ utra,uitr,uwal
        common /cstr/ sxx,syy,szz
c
c        common blocks
c
         common /cord/ x,y,z,u,v,w,nq
         common /cvef/ ve,vf,die,dif
         common /ctab/ lr,g
         common /cpar/ xmax,xmax2,rm1,rm2,rm3,rp1,rp2,rp3,dt
         ep=0.
         etot=0.
         do 20 j=1,np
           ax(j)=0.
           ay(j)=0.
           az(j)=0.
20       continue
           e2=0.
           e3=0.
           v2=0.
           v3=0.
         do 200 j=1,np,nsite
c           write(6,101)
c101        format(' call force2')
            call force2(fx,fy,fz,ct,e2,e3,cphi,j)
           do 190 kk=j,j+(nsite-3)
                ix=(ct(kk)+1.)*50+1
                if(ix.gt.500) ix=500
                if(ix.lt.0) ix=498
                g(2,ix)=g(2,ix)+1.
190        continue
            do 192 kk=j,j+(nsite-4)
                ix=(cphi(kk)+1.)*50 +101
                g(2,ix)=g(2,ix)+1.
                g(2,499)=g(2,499)+1.
192         continue
              do 95 jk=j,j+(nsite-1)
                ax(jk)=ax(jk)+fx(jk)
                ay(jk)=ay(jk)+fy(jk)
                az(jk)=az(jk)+fz(jk)
95            continue
200       continue
           ep=ep+e2+e3
          nx=1
          do 50 j=1,np-1
c         Begin njk determination
           mj=mod(j,nsite)
            if(mj.gt.1) then
               mj=2
            else if(mj.eq.1) then
               mj=0
            else
               mj=3
            end if
30           k=lr(nx)
            nx=nx+1
            if(k.eq.0) go to 50
            if(nq(k).eq.nq(j)) then
c            This point should NEVER be reached.
             write(6,201)
201          format(' TROUBLE with intramolecular counting!')
             stop
            else
c         Finish up determination of njk-Note this else goes to 45 cont.
           mk=mod(k,nsite)
            if(mk.gt.1) then
               mk=2
            else if(mj.eq.1) then
               mk=0
            else
               mk=3
            end if
            njk=mj+mk
             if(njk.eq.0) then njk=1
c           This completes the determination of njk
             xx=x(j)-x(k)
             yy=y(j)-y(k)
             zz=z(j)-z(k)
             if(xx.gt.xmax2)  xx=xx-xmax
             if(xx.lt.-xmax2) xx=xx+xmax
             if(yy.gt.xmax2)  yy=yy-xmax
             if(yy.lt.-xmax2) yy=yy+xmax
             rr=xx*xx+yy*yy+zz*zz
               if(rr.gt.rm1) go to 45
             rr=sqrt(rr)
```

```
              r=100.*rr
              if(r.lt.30) go to 60
              l=int(r)
              alpha=r-l
              phi=ve(njk,l)+alpha*die(njk,l)
              f=vf(njk,l)+alpha*dif(njk,l)
            qk=f/rr
               qjx=qk*xx
               ax(j)=ax(j)+qjx
               ax(k)=ax(k)-qjx
               qjy=qk*yy
               ay(j)=ay(j)+qjy
               ay(k)=ay(k)-qjy
               qjz=qk*zz
               az(j)=az(j)+qjz
               az(k)=az(k)-qjz
            ep=ep+phi
             jj=100*z(j)
             jk=100*z(k)
              sxx(jj)=sxx(jj)+0.5*qjx*xx
              sxx(jk)=sxx(jk)+0.5*qjx*xx
              syy(jj)=syy(jj)+0.5*qjy*yy
              syy(jk)=syy(jk)+0.5*qjy*yy
              szz(jj)=szz(jj)+0.5*qjz*zz
              szz(jk)=szz(jk)+0.5*qjz*zz
              uitr(jj)=uitr(jj)+0.5*phi
              uitr(jk)=uitr(jk)+0.5*phi
          end if
45        go to 30
50        continue
          etot=ep/(1.*np)
          return
60        write(6,1) r,j,k
1         format(' r =',e10.4,2i5)
          write(6,2) x(j),y(j),z(j)
          write(6,2) x(k),y(k),z(k)
2         format(3e15.5)
          stop 'Range trouble'
          end
      subroutine force2(fx,fy,fz,ct,e2,e3,cphi,jp)
c     Forces  checked and corrected  7-29-97  RDM
       parameter (nsite=8, nmols=225, np=nsite*nmols)
      dimension x(np),y(np),z(np),fx(np),fy(np),fz(np),u(np),v(np),
     x   w(np),nq(np),ct(np),hx(np),hy(np),hz(np)
      dimension cx(np),cy(np),cz(np),r(np),gx(np),gy(np),gz(np)
      dimension rr(np),ex(np),ey(np),ez(np),cphi(np)
       dimension ve(9,800),vf(9,800),die(9,800),dif(9,800)
      double precision sxx(0:1000),syy(0:1000),szz(0:1000)
      double precision utra(0:1000),uitr(0:1000),uwal(0:1000)
      common /cuuu/ utra,uitr,uwal
      common /cstr/ sxx,syy,szz
       common /cord/ x,y,z,u,v,w,nq
       common /cvef/ ve,vf,die,dif
       common /cpar/ xmax,xmax2,rm1,rm2,rm3,rp1,rp2,rp3,dt
       fscale= 57803.
```

```
         d0=1.53/3.16
         fs3=798.8
         pi=4.*atan(1.)
         ct0=114.00*pi/180.
         ct0=cos(ct0)
         a0=1116./78.24
         a1=1462./78.24
         a2=-1578./78.24
         a3=-368./78.24
         a4=3156./78.24
         a5=-3788./78.24
        do 5 j=jp,jp+(nsite-1)
         fx(j)=0.
         fy(j)=0.
         fz(j)=0.
         gx(j)=0.
         gy(j)=0.
         gz(j)=0.
         hx(j)=0.
         hy(j)=0.
         hz(j)=0.
5       continue
c       Stretch
        do 10 j=jp,jp+(nsite-2)
          cx(j)=x(j+1)-x(j)
          cy(j)=y(j+1)-y(j)
          cz(j)=z(j+1)-z(j)
c       Check the periodic boundary conditions!
          if(cx(j).gt.xmax2) then
             cx(j)=cx(j)-xmax
          else if(cx(j).lt.-xmax2) then
             cx(j)=cx(j)+xmax
          end if
          if(cy(j).gt.xmax2) then
             cy(j)=cy(j)-xmax
          else if(cy(j).lt.-xmax2) then
             cy(j)=cy(j)+xmax
          end if
ccc     Not for the z-components!
c          if(cz(j).gt.xmax2) then
c             cz(j)=cz(j)-xmax
c          else if(cz(j).lt.-xmax2) then
c             cz(j)=cz(j)+xmax
c          end if
         rr(j)=cx(j)**2+cy(j)**2+cz(j)**2
         r(j)=sqrt(rr(j))
          ex(j)=cx(j)/r(j)
          ey(j)=cy(j)/r(j)
          ez(j)=cz(j)/r(j)
         f=-fscale*(r(j)-d0)
         fx(j+1)=fx(j+1)+f*ex(j)
         fy(j+1)=fy(j+1)+f*ey(j)
         fz(j+1)=fz(j+1)+f*ez(j)
          fx(j)=fx(j)-f*ex(j)
         fy(j)=fy(j)-f*ey(j)
         fz(j)=fz(j)-f*ez(j)
        e2=e2+0.5*fscale*(r(j)-d0)**2
c       Pressure tensor components:
         ji=100*z(j)
         jk=100*z(j+1)
            ff=0.5*f*ex(j)*cx(j)
         sxx(ji)=sxx(ji)+ff
         sxx(jk)=sxx(jk)+ff
            ff=0.5*f*ey(j)*cy(j)
         syy(ji)=syy(ji)+ff
         syy(jk)=syy(jk)+ff
            ff=0.5*f*ez(j)*cz(j)
         szz(ji)=szz(ji)+ff
         szz(jk)=szz(jk)+ff
         utra(ji)=utra(ji)+0.25*fscale*(r(j)-d0)**2
         utra(jk)=utra(jk)+0.25*fscale*(r(j)-d0)**2
10     continue
c         write(6,9999) ct0
c9999     format(' ct0 =',e10.3)
       do 20 j=jp,jp+(nsite-3)
        ct(j)=-(ex(j)*ex(j+1)+ey(j)*ey(j+1)+ez(j)*ez(j+1))
c          theta=acos(ct(j))
c        f3=-fs3*(theta-ct0)*(-1./sqrt(1.-ct(j)*ct(j)))
         f3=-fs3*(ct(j)-ct0)
        ax=f3/r(j)*(ex(j+1)+ct(j)*ex(j))
        bx=f3/r(j+1)*(-ex(j)-ct(j)*ex(j+1))
         gx(j)=  gx(j)  +ax
         gx(j+2)=gx(j+2)+bx
         gx(j+1)=gx(j+1)-ax-bx
        ay=f3/r(j)*(ey(j+1)+ct(j)*ey(j))
        by=f3/r(j+1)*(-ey(j)-ct(j)*ey(j+1))
         gy(j)=  gy(j)  +ay
         gy(j+2)=gy(j+2)+by
         gy(j+1)=gy(j+1)-ay-by
        az=f3/r(j)*(ez(j+1)+ct(j)*ez(j))
        bz=f3/r(j+1)*(-ez(j)-ct(j)*ez(j+1))
         gz(j)=  gz(j)  +az
         gz(j+2)=gz(j+2)+bz
         gz(j+1)=gz(j+1)-az-bz
        e3=e3+0.5*fs3*(ct(j)-ct0)**2
c       Pressure components, bond-by-bond
          ji=100*z(j)
          jj=100*z(j+1)
          jk=100*z(j+2)
            gg=0.5*ax*cx(j)
             sxx(ji)=sxx(ji)-gg
             sxx(jj)=sxx(jj)-gg
            gg=0.5*ay*cy(j)
             syy(ji)=syy(ji)-gg
             syy(jj)=syy(jj)-gg
            gg=0.5*az*cz(j)
             szz(ji)=szz(ji)-gg
            szz(jj)=szz(jj)-gg
c      Now the other bond
           gg=0.5*bx*cx(j+1)
            sxx(jj)=sxx(jj)+gg
            sxx(jk)=sxx(jk)+gg
           gg=0.5*by*cy(j+1)
            syy(jj)=syy(jj)+gg
            syy(jk)=syy(jk)+gg
           gg=0.5*bz*cz(j+1)
            szz(jj)=szz(jj)+gg
            szz(jk)=szz(jk)+gg
          utra(ji)=utra(ji)+0.5/3.*fs3*(ct(j)-ct0)**2
          utra(jj)=utra(jj)+0.5/3.*fs3*(ct(j)-ct0)**2
          utra(jk)=utra(jk)+0.5/3.*fs3*(ct(j)-ct0)**2
20    continue
c     Twist or torsion 4-molecules per unit considered.
      do 30 j=jp,jp+(nsite-4)
       ct1=cx(j)*cx(j+1)+cy(j)*cy(j+1)+cz(j)*cz(j+1)
       ct2=cx(j+1)*cx(j+2)+cy(j+1)*cy(j+2)+cz(j+1)*cz(j+2)
       ct3=cx(j)*cx(j+2)+cy(j)*cy(j+2)+cz(j)*cz(j+2)
        d1=rr(j  )*rr(j+1)-ct1*ct1
        d2=rr(j+1)*rr(j+2)-ct2*ct2
         d3=sqrt(d1*d2)
       cphi(j)=(rr(j+1)*ct3-ct1*ct2)/d3
       fs4=a1+2.*a2*cphi(j)+3.*a3*cphi(j)**2+4.*a4*cphi(j)**3+
     x       5.*a5*cphi(j)**4
        ex4=a0+a1*cphi(j)+a2*cphi(j)**2+a3*cphi(j)**3+a4*cphi(j)**4+
     x        a5*cphi(j)**5
      e3=e3+ex4
c     Generate the bond-derivatives of the energy.
c       ji-->j, kj-->j+1, lk-->j+2
c      First the ji-bond
       ujix=    (-cx(j+1)*ct2    +cx(j+2)*rr(j+1))/d3
     x -cphi(j)*( cx(j  )*rr(j+1)-cx(j+1)*ct1    )/d1
       ujiy=    (-cy(j+1)*ct2    +cy(j+2)*rr(j+1))/d3
     y -cphi(j)*( cy(j  )*rr(j+1)-cy(j+1)*ct1    )/d1
       ujiz=    (-cz(j+1)*ct2    +cz(j+2)*rr(j+1))/d3
     z -cphi(j)*( cz(j  )*rr(j+1)-cz(j+1)*ct1    )/d1
c      Second the kj-bond
       ukjx=(-cx(j)*ct2-cx(j+2)*ct1+2.*cx(j+1)*ct3)/d3
     x  -cphi(j)*((cx(j+1)*rr(j  )-cx(j  )*ct1)/d1
     x           +(cx(j+1)*rr(j+2)-cx(j+2)*ct2)/d2)
       ukjy=(-cy(j)*ct2-cy(j+2)*ct1+2.*cy(j+1)*ct3)/d3
     y  -cphi(j)*((cy(j+1)*rr(j  )-cy(j  )*ct1)/d1
     y           +(cy(j+1)*rr(j+2)-cy(j+2)*ct2)/d2)
       ukjz=(-cz(j)*ct2-cz(j+2)*ct1+2.*cz(j+1)*ct3)/d3
     z  -cphi(j)*((cz(j+1)*rr(j  )-cz(j  )*ct1)/d1
     z           +(cz(j+1)*rr(j+2)-cz(j+2)*ct2)/d2)
c      Finally the lk-bond
       ulkx=(-cx(j+1)*ct1+cx(j)*rr(j+1))/d3
     x -cphi(j)*(cx(j+2)*rr(j+1)-cx(j+1)*ct2)/d2
       ulky=(-cy(j+1)*ct1+cy(j)*rr(j+1))/d3
     y -cphi(j)*(cy(j+2)*rr(j+1)-cy(j+1)*ct2)/d2
```

```
        ulkz=(-cz(j+1)*ct1+cz(j)*rr(j+1))/d3
     z -cphi(j)*(cz(j+2)*rr(j+1)-cz(j+1)*ct2)/d2
c      Now construct the forces and the pressure components
        hx(j)=  hx(j  )+fs4*  ujix
        hx(j+1)=hx(j+1)+fs4*(-ujix+ukjx)
        hx(j+2)=hx(j+2)+fs4*(-ukjx+ulkx)
        hx(j+3)=hx(j+3)+fs4*(-ulkx)
        hy(j)=  hy(j  )+fs4*  ujiy
        hy(j+1)=hy(j+1)+fs4*(-ujiy+ukjy)
        hy(j+2)=hy(j+2)+fs4*(-ukjy+ulky)
        hy(j+3)=hy(j+3)+fs4*(-ulky)
        hz(j)=  hz(j  )+fs4*  ujiz
        hz(j+1)=hz(j+1)+fs4*(-ujiz+ukjz)
        hz(j+2)=hz(j+2)+fs4*(-ukjz+ulkz)
        hz(j+3)=hz(j+3)+fs4*(-ulkz)
        ji=100*z(j)
        jj=100*z(j+1)
        jk=100*z(j+2)
        jl=100*z(j+3)
c       The ji-bond terms
         sxx(ji)=sxx(ji)-0.5*fs4*ujix*cx(j)
         sxx(jj)=sxx(jj)-0.5*fs4*ujix*cx(j)
         syy(ji)=syy(ji)-0.5*fs4*ujiy*cy(j)
         syy(jj)=syy(jj)-0.5*fs4*ujiy*cy(j)
         szz(ji)=szz(ji)-0.5*fs4*ujiz*cz(j)
         szz(jj)=szz(jj)-0.5*fs4*ujiz*cz(j)
c       The kj-bond terms
         sxx(jj)=sxx(jj)-0.5*fs4*ukjx*cx(j+1)
         sxx(jk)=sxx(jk)-0.5*fs4*ukjx*cx(j+1)
         syy(jj)=syy(jj)-0.5*fs4*ukjy*cy(j+1)
         syy(jk)=syy(jk)-0.5*fs4*ukjy*cy(j+1)
         szz(jj)=szz(jj)-0.5*fs4*ukjz*cz(j+1)
         szz(jk)=szz(jk)-0.5*fs4*ukjz*cz(j+1)
c       The lk-bond terms
         sxx(jk)=sxx(jk)-0.5*fs4*ulkx*cx(j+2)
         sxx(jl)=sxx(jl)-0.5*fs4*ulkx*cx(j+2)
         syy(jk)=syy(jk)-0.5*fs4*ulky*cy(j+2)
         syy(jl)=syy(jl)-0.5*fs4*ulky*cy(j+2)
         szz(jk)=szz(jk)-0.5*fs4*ulkz*cz(j+2)
         szz(jl)=szz(jl)-0.5*fs4*ulkz*cz(j+2)
        utra(ji)=utra(ji)+ex4/4.
        utra(jj)=utra(jj)+ex4/4.
        utra(jk)=utra(jk)+ex4/4.
        utra(jl)=utra(jl)+ex4/4.
30     continue
       do 50 j=jp,jp+(nsite-1)
        fx(j)=fx(j)+gx(j)+hx(j)
        fy(j)=fy(j)+gy(j)+hy(j)
        fz(j)=fz(j)+gz(j)+hz(j)
50     continue
c      Lennard-Jones interactions between "distant" sites
       do 60 j=jp,jp+(nsite-5)
         mj=mod(j,nsite)
```

```
          if(mj.gt.1) then
             mj=2
          else if(mj.eq.1) then
             mj=0
          else
             mj=3
          end if
        do 55 k=j+4,jp+(nsite-1)
        dx=x(j)-x(k)
         if(dx.gt.xmax2) then
            dx=dx-xmax
         else if(dx.lt.-xmax2) then
            dx=dx+xmax
         end if
        dy=y(j)-y(k)
         if(dy.gt.xmax2) then
            dy=dy-xmax
         else if(dy.lt.-xmax2) then
            dy=dy+xmax
         end if
        dz=z(j)-z(k)
        rq=dx*dx+dy*dy+dz*dz
         if(rq.gt.rm1) go to 60
         mk=mod(k,nsite)
          if(mk.gt.1) then
             mk=2
          else if(mk.eq.1) then
             mk=0
          else
             mk=3
          end if
         njk=mj+mk
           if(njk.eq.0) then njk=1
c         This completes the determination of njk
        rq=sqrt(dx*dx+dy*dy+dz*dz)
        rp=100.*rq
        l=int(rp)
        alpha=rp-l
        phi=ve(njk,l)+alpha*die(njk,l)
        ff= (vf(njk,l)+alpha*dif(njk,l))/rq
        fx(j)=fx(j)+ff*dx
        fx(k)=fx(k)-ff*dx
        fy(j)=fy(j)+ff*dy
        fy(k)=fy(k)-ff*dy
        fz(j)=fz(j)+ff*dz
        fz(k)=fz(k)-ff*dz
        e3=e3+phi
c      Next the pressure tensor terms
        ji=100*z(j)
        jl=100*z(k)
        sxx(ji)=sxx(ji)+0.5*ff*dx*dx
        sxx(jl)=sxx(jl)+0.5*ff*dx*dx
        syy(ji)=syy(ji)+0.5*ff*dy*dy
        syy(jl)=syy(jl)+0.5*ff*dy*dy
        szz(ji)=szz(ji)+0.5*ff*dz*dz
        szz(jl)=szz(jl)+0.5*ff*dz*dz
```

```
       utra(ji)=utra(ji)+0.5*phi
       utra(jl)=utra(jl)+0.5*phi
55     continue
60    continue
      do 70 j=jp,jp+(nsite-1)
        rq=z(j)
        rp=100*rq
        l=int(rp)
        alpha=rp-l
        mj=mod(j,nsite)
        if(mj.gt.1) then
           mj=8
        else if(mj.eq.1) then
           mj=7
        else
           mj=9
        end if
          phi=ve(mj,l)+alpha*die(mj,l)
          ff=vf(mj,l)+alpha*dif(mj,l)
        fz(j)=fz(j)+ff
        e3=e3+phi
        jj=100*z(j)
          szz(jj)=szz(jj)+ff*z(j)
        uwal(jj)=uwal(jj)+phi
70    continue
      return
      end
```